

Lezione 13 - Valutazione delle Prestazioni

- Le prestazioni sono in unità di "oggetti-per-secondo"

- "grande" è meglio

Vorrei un solo numero che mi indichi le prestazioni della macchina

- Se siamo soprattutto interessati al tempo di risposta

- $P_x = \frac{1}{E_x}$ più piccolo è e meglio è

Prestos. macchina x = $\frac{1}{\text{Tempo di esecuzione del programma}}$

- P_x = prestazioni della macchina X

- E_x = tempo di esecuzione della macchina X *del programma*

Dipende dal programma e meglio confrontare

- Speed up rispetto a una macchina di riferimento

- $S_{xy} = \frac{P_x}{P_y} = \frac{E_y}{E_x}$ *rapporto tempi esecuzione, se > 1 X è più veloce rispetto a Y*

- Lo speed up di X rispetto a Y dice "X è S_{xy} volte più veloce di Y"

- In percentuali, $s_{xy} = (S_{xy} - 1) * 100$ dice "X è $s_{xy}\%$ più veloce di Y" *qual è la macchina di riferimento?*

- Esempio:

- $E_x = 1\text{sec}, E_y = 1.2\text{sec} \Rightarrow S_{xy} = 1.2 \Rightarrow X \text{ è } 1.2 \text{ volte più veloce di Y}$
 $s_{xy} = 20\% \Rightarrow X \text{ è il } 20\% \text{ più veloce di Y}$

qual è la macchina di riferimento?

Qual è il tempo di esecuzione del programma? cosa misura? da cosa dipende?

Tempo Totale e Tempo di CPU

uso come r. perimetro il tempo

• T_J = Tempo Totale *Pos includere tempi dovuti a ritardi esterni* *Tempi che dipendono da ciò che succede*
(o Tempo Assoluto o Tempo di Risposta o Tempo Trascorso (Elapsed Time))

• tempo necessario per completare un lavoro; comprende una serie di tempi che sono "allungati" dalle interferenze di altri programmi che generano attività come:

- accesso ai dischi
- accesso alla memoria
- attività di I/O
- sovraccarico del Sistema Operativo

} *dipendono anche da altri ritardi esterni*

Quantità più precisa *Filtrando i tempi aggiuntivi, come I/O ecc*
• T_{CPU} = Tempo di CPU (Tempo di Esecuzione del programma) *eliminando*

• Tempo durante il quale un processore ha lavorato su un singolo programma

- senza tempo speso in I/O (può includere il tempo "dell'uomo")
- senza tempo per eseguire altri programmi

Tempi spesi per I/O ecc

• Può essere suddiviso in

- T_U = tempo di CPU relativo all'utente *al suo programma*
- T_K = tempo di CPU relativo al Sistema Operativo (K=Kernel) *es. sys. op.*

macchina non può fare a meno del S.O.

può far a meno del Kernel (sys op.)

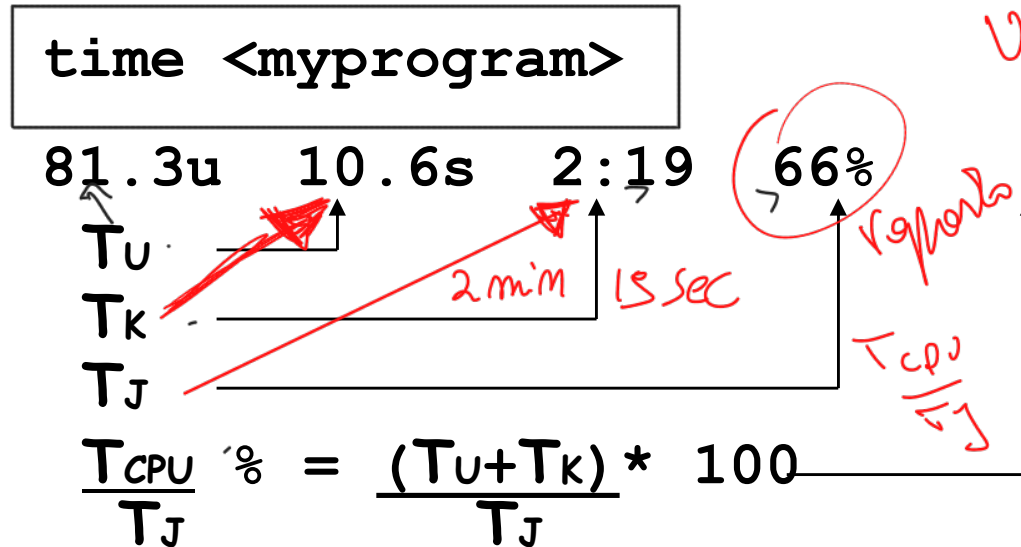
$$\Rightarrow T_U + T_K = T_{CPU}$$

istruzioni eseguite dal programma mostrato

Tempo speso dal sistema operativo per eseguire richieste del processo

Tempo di CPU misurato in UNIX/LINUX

- Dato il programma myprogram



$U = \text{utente}$

$S = \text{system, kernel}$

$T_j = \text{Tempo Job}$

Per il 34 % la macchina è usata

- per I/O
- per altri programmi
- per altra attività di sistema

- TK spesso non viene incluso
 - perché può nascondere inaccuratezze di misura
 - il calcolo può risultare sbilanciato se si usano sistemi operativi diversi
 - su alcune macchine il codice può essere considerato di kernel e su altre di utente
 - d'altra parte nessun programma può essere eseguito senza usare qualche parte di kernel

- Prestazioni di Sistema $\Rightarrow T_J$ *del sistema in generale, Time Job*

Prestazioni CPU $\Rightarrow T_U$ *Tempo utente*
Tempo CPU per risolvere il mio lavoro

escludendo servizi e programmi S.O.

Equazione delle prestazioni

Cosa influenza il T_{CPU} ? la macchina ha un certo clock

Tempo esecuzione programma

$$T_{CPU} = C_{CPU} \cdot T_C = \frac{C_{CPU}}{f_c} = \underbrace{N_{CPU} \cdot \overline{CPI}}_{\substack{\text{N istruzioni eseguite} \\ \text{DINAMICAMENTE}}} \cdot T_C$$

cicli iniezione
periodo clock
istruzioni
ogni ciclo di clock viene eseguito qualcosa
N istruzioni eseguite DINAMICAMENTE
N cicli per eseguire quelle intr.

- T_C è il periodo di clock (f_c è la frequenza di clock)
- C_{CPU} sono i cicli di clock
- N_{CPU} è il Numero di istruzioni eseguite **DINAMICAMENTE** *può essere diverso*
- $\overline{CPI}$ è il numero di Cicli Per Istruzione (medio) *es. Ho eseguito 3500 istruzioni, una medialemp inizia 2, 3 cicli*

Esempio:

bne → 2 cicli
 add → 1 ciclo
 2 istruzioni, 3 cicli → $\overline{CPI} = 3/2 = 1.5$

cicli per istruzioni

... C_{CPU} cicli per eseguire il comando

Esempio 1

Assumiamo $t0=0$, $t1=0$:

2 cicli `addi t1, t1, 100`

`loop: add t0, t0, 1`

`bne t0, t1, loop` *loop eseguita 100 volte*

ADD richiede 2 cicli di clock, BNE 2

*STATICAM. Sono 3 istr.
DYNAMIC. 100*

*#201 istruzioni
Add, 2 ciclo
BNE = 3 cicli*



Cicli di CPU $C_{CPU} = 502$, $N_{CPU} = 201$ *# istruzione
numero dinamico*

$CPI = 502/201 \approx 2.5$

$$CPI = \frac{502}{201} \approx 2,5$$

ADD, BNE, 100 volte

• Per ottenere maggiori prestazioni si può:

• ridurre i cicli di clock per un programma

• aumentare la frequenza di clock

• Spesso però l'abbassamento del numero di cicli

comporta invece un **abbassamento** della frequenza di clock

meno cicli per eseg. un programma

*più alto è il clock, più consumo, non accede più
aumentare la freq.*

più energia se lo clock

Esempio 2

$$T_{CPU} = \frac{C_{CPU}}{f} = 10 \quad C_{pi} = 4000$$

- $T_{CPU}(A) = 10s$ $f_c(A) = 400 \text{ MHz}$

- $T_{CPU}(B) = 6s$ $f_c(B) = ?$

← **MACCHINA A**

← **MACCHINA B**

$$T_{CPU} = \frac{4000}{f} = 6 \Rightarrow f = \frac{4000}{6} = 666,66 \text{ MHz}$$

Supponiamo che in B si riesca a ridurre il tempo ma con il risultato di avere

$$666,66 \cdot 1,2$$

$$C_{CPU}(B) = 1.2 * C_{CPU}(A)$$

- Quanto deve essere la frequenza di clock della macchina B affinché il tempo di esecuzione del programma sia 6s?

Soluzione:

$$C_{CPU}(A) = (f_c * T_{CPU})|_A = 4 * 10^9$$

$$f_c(B) = (C_{CPU} / T_{CPU})|_B = (1.2 * 4 * 10^9 / 6) = 800 \text{ MHz}$$

- Per ottenere una diminuzione del 40% del tempo di esecuzione è necessario raddoppiare la frequenza di clock e consumare di più
invece del 40%,

Esempio 3

$$CPI_A = \frac{2.5 \cdot N_{CPUA}}{1 \cdot 10^9} = 2.5 \times 10^{-9}$$

$$T_{CPUB} = \frac{1.2 \cdot N_{CPUB}}{0.5 \cdot 10^9} = 2.4 \cdot 10^{-9}$$

- $f_c(A) = 1 \text{ GHz}$ $f_c(B) = 500 \text{ MHz}$
- $/CPI(A) = 2.5$ $/CPI(B) = 1.2$
- Quale calcolatore è più veloce?
(il set di istruzioni rimane lo stesso)

-
- $S_{AB} = \frac{T_{CPU}(B)}{T_{CPU}(A)} = \frac{N_{CPU}(B) \cdot /CPI(B) \cdot f_c(A)}{N_{CPU}(A) \cdot /CPI(A) \cdot f_c(B)} = \frac{1.2 \cdot 1000}{2.5 \cdot 500}$
 - set istruzioni uguale $\rightarrow N_{CPU}(A) = N_{CPU}(B)$
 - $S_{AB} = 0.9 \rightarrow B$ è 1.1 volte più veloce di A

NCPU dipende da

NCU dipende da almeno 4 cose

Motivi delle dipendenze dei parametri fondamentali

#iow.

- Programma dipende dal programmatore, che algoritmo usare *ovvero # di istruzioni eseguite di volta*
 - Fibonacci-ricorsivo vs. Fibonacci-iterativo → N_{CPU}

- Compilatore *tipo ottimizzatore*, sceglie quale operaz. usare per fare una cosa
 - Opzioni -O1, -O2, -O3 → N_{CPU} , /CPI *compilatori offrono grado di ottimizzazione influenza # istruzioni eseguite, una anche il conteggio cicli*

- Sistema Operativo
 - Una syscall può essere implementata diversamente in Linux rispetto a Windows → N_{CPU} influenza il conteggio delle istruz.
- Instruction Set (ISA)
 - RISC-V vs. x86 vs. ARM, #cicli necessari per es. add → N_{CPU} , /CPI *ADD 1 ciclo vs. Add 3 cicli*

dipende da ✓	NCPU	/CPI	fc
Programma come è scritto	X		
Compilatore e sua scelta di usare	X	(X)	
Sistema Operativo e call	X		
ISA (Instruction set)	X	X	
Struttura CPU		X	X
Sottosistema di Memoria		X	
Tecnologia			X

TCPU	→ orologio
fc	→ dato di targa
NCPU, /CPI	→ difficili da misurare

- Struttura CPU
 - Una pipeline elabora più istruzioni per ciclo → /CPI *ci si può migliorare*
 - Stadi (logica combinatoria) con meno porte logiche → maggiore fc

- Sottosistema di memoria
 - Es. add → 1 ciclo, lw → 100 cicli *MEM. DATA, CACHE...*

- Tecnologia (elettronica)
 - Es. 14 nm invece che 90nm *distinzione tra transistor in termini memoria*

or.
ADD richiede 3 cicli in x86 e 1 in RISC-V

clock può aumentare, più cose in corso contemporaneamente, più logica è veloce

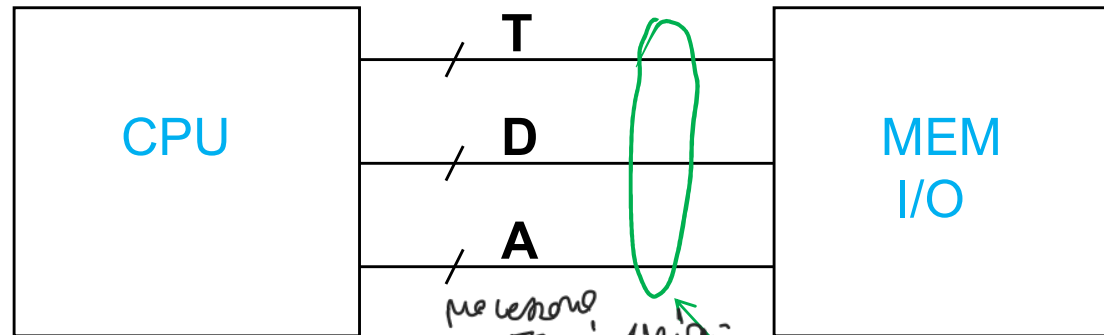
$$T_c = \frac{1}{fc}$$

es. Mem. Mem. più grande in cache
una load costa 100 volte di più

chip più piccoli, freq. manipolati

Come valutare Ncpu (1) - Sonda HW

Strumenti per monitorare Ncpu



Il proc. prende i dati, scambia dati

T = TYPE = {IF, DR, DW}

FETCH DATA-READ DATA-WRITE

D = DATA

processore e dati

A = ADDRESS

indirizzi,
scambia dati
e scambia tipo
di operazioni
(FETCH)

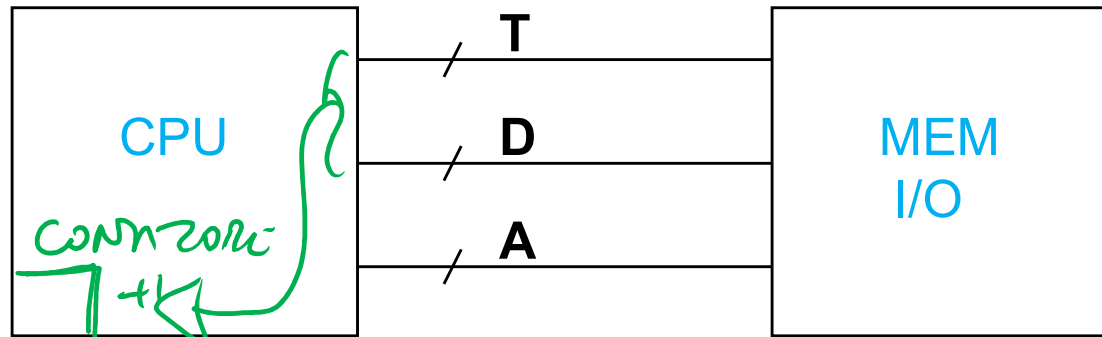
memoria
e dati indirizzi
e scambio
dati

Sonda analizzatrice a
stati logici

- Strumentazione Hardware (sonda o probe) che legge le istruzioni che il processore preleva dalla memoria
- Richiede di poter aprire la macchina
- Richiede la disponibilità di una opportuna sonda
- Richiede la disponibilità di un analizzatore di stati logici

Come valutare Ncpu (2) - Contatori HW

registri nascosti dove effettua
conteggi, r-es.



- marker nel programma

Conteggi o operaz.
eseguite

T = TYPE = {IF, DR, DW}
FETCH DATA-READ DATA-WRITE

D = DATA

A = ADDRESS

• Contatori Hardware che contano il numero di istruzioni

- Disponibili sui processori Intel (ed Alpha)

• Supporto da parte del processore e del sistema

- Alpha: ATOM tools, Intel VTUNE

Come valutare Ncpu (3) - Instrumentazione SW (Sonda-SW)

Codice Originale

```
add t0,t1,t2    #1ciclo
lw  t0,0(t1)    #100cicli
bne t3,t4,Loop  #3cicli
```

Codice Instrumentation

```
add s8,zero,3
add s9,zero,104
add t0,t1,t2
lw  t0,0(t1)
bne t3,t4,Loop
```

*inserisco
conteggio istru.
e conteggio
di Ncicli*

- 1) Riservo un registro (es. **s8**) per contare le istruzioni *es. s8*
- 2) Riservo un registro (es. **s9**) per contare i cicli di ogni basic-block* *es. s9*
- 3) Inserisco due istruzioni per tali conteggi per ogni basic-block* del programma (instrumentazioni)
- 4) Al termine del programma, s8 contiene Ncpu mentre s9/s8 è pari a /CPI
cicli / istru.

• Metodo usato in vari processori

• MIPS R4400: Pixie Alpha: Atom Intel: VTUNE

*- Isten. codice quale è il
basic block più presente
ecc., quale da istruire ecc.*

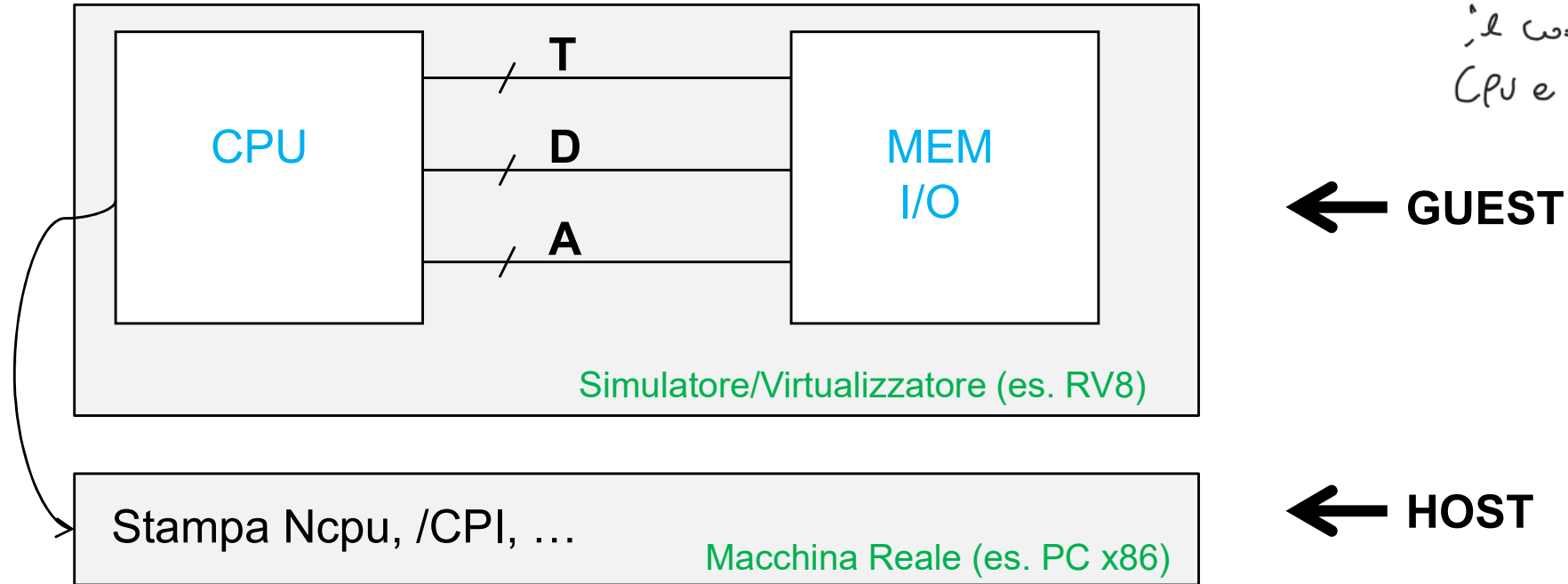


*Cicli necessari per program
se non ci fossero le 2 istru.*

*Basic-block=gruppo di istruzioni con un solo punto di ingresso e un solo punto di uscita

Come valutare Ncpu (4) - Virtual-Machine SW

*è SW, può girare
il codice e interfaccia
CPU e I/O*



• Usare dei simulatori/virtualizzatori/emulatori

*Passo allora viene accennato
alla CPU*

- RV8 → simulatore di processore RISC-V
- SPIM → simulatore di processore MIPS
- QEMU → emulatore molto diffuso sotto Linux → virtual machine
- Virtualbox, Vmware, VirtualPC → ulteriori virtual machines

Come calcolare /CPI

clock medio per instr.

$$\overline{CPI} = \frac{\sum_{k=1}^{T} N_{CPU,k} \cdot CPI_k}{N_{CPU}}$$

Si può calcolare da quante istruzioni di ogni tipo vengono eseguite ($N_{CPU,k}$)

$$CICLI = \overset{\# \text{ instr.}}{N_{CPU}} \cdot \overline{CPI}$$

$$CICLI = \sum_{K=1}^T N_{CPU,K} \cdot CPI_K$$

$$\overline{CPI} = \sum_{k=1}^T \overline{CPI}_k = \sum_{k=1}^T p_{CPU,k} \cdot CPI_k = \sum_{k=1}^T \frac{N_{CPU,k}}{N_{CPU}} \cdot CPI_k$$

divido per i vari tipi di istruzioni
percentuale instr. tipo k che ci vuole
di quel tipo k
instr. eseguite

- T = numero tipi diversi di istruzioni
 $N_{CPU,k}$ = numero di istruzioni di tipo K
 $p_{CPU,k}$ = frequenza relativa di quel tipo di istruzione $\equiv N_{CPU,k}/N_{CPU}$
 CPI_K = CPI delle istruzioni di tipo K
 $/CPI_K$ = CPI **MEDIO** delle istruzioni di tipo K $\equiv p_{CPU,k} * CPI_K$

T = tutti i tipi di instr. diversi

Op	$p_{CPU,k}$ $\equiv N_{CPU,k}/N_{CPU}$	CPI_K	$/CPI_K$ $\equiv p_{CPU,k} * CPI_K$	$\%T_{CPU,k} = CICLI_k / CICLI$ $/CPI_K : /CPI$
Arithmetic	0.50 <i>50% - load store instr.</i>	1 <i>prop. aritmetica</i>	0.5	33% <i>impiega il 33% dei cicli cioè del Tempo</i>
Load	0.20	2	0.4	27%
Store	0.10	2	0.2	13%
Branch	0.20	2 <i>es. 2 cicli per fare Branch</i>	0.4	27%
$/CPI = 0.5 + 0.4 + 0.2 + 0.4 = 1.5$				

Tempo in cui di spesso più lo step instr.

Scalabilità

Capacità di un sistema a seguire un'evoluzione rispetto ad un parametro, es. freq. con la freq.

$P := Prestazioni$

$$P \propto 1/T_{CPU} = \frac{1}{N_{CPU} \cdot CPI} \cdot f_C$$

Probs. inters. per tempo (green)
Prest. scalabili (orange)
se raddopp. freq. (orange)
raddoppio prest. (orange)
per Teo. (green)

- Diciamo che un sistema ha **prestazioni scalabili con la frequenza** se, con $f_{200}/f_{100}=2$, ottengo $P_{200}/P_{100}=2$ es. raddoppiando f, raddoppio prest.
- Abbiamo visto però che, aumentando la frequenza, è verosimile che il CPI aumenti. es. $CPI_{200}=1.2 \cdot CPI_{100}$; in tal caso si otterrebbe $P_{200}/P_{100}=f_{200}/f_{100} \cdot CPI_{100}/CPI_{200}=2/1.2 \approx 1.6$ quindi poca scalabilità (<2) raddoppio clock, con lo stesso velocità (orange)
- Questo è ciò che si è verificato storicamente nei Pentium
 - Per es., se il sottosistema di memoria 'non tiene il passo del processore' architettura superscalare
 - Risolto nei PentiumPro mantenendo uno stesso CPI all'aumentare di f_C
- La capacità di fornire maggiori prestazioni al variare di un parametro viene chiamata **scalabilità rispetto a quel parametro**
- Per es., il processore PentiumPro risulta più scalabile del processore Pentium con la frequenza ($Scalabilità(f)_{PentiumPro} > Scalabilità(f)_{Pentium}$)

- Diciamo che un sistema ha **scalabilità ideale lineare** se $Scalabilità(p) = k \cdot p$ se scalabilità aumenta linearmente rispetto a quel dove (k=costante, p=parametro)
- scalabilità rispetto ad un parametro, freq. (green)

ci dice come può dipendere una parte del sist rispetto tutto il sist



Legge di Amdhal

quali sono i fattori limitanti

analisi speedup del sistema

- In generale può essere difficile stabilire l'esatta dipendenza da un dato parametro (anche quando questo compare esplicitamente nell'equazione!)

Come può dipendere una parte del sistema da tutto il sist

Analisi Speed-up sistema quando ne miglioro una parte

- Per questo motivo viene utilizzata la seguente formula, per mettere in relazione lo speedup di una parte del sistema con lo speedup dell'intero sistema:

$$S = \frac{P_{dopo}}{P_{prima}} = \frac{T_{prima}}{T_{dopo}} = \frac{T_{prima}}{\frac{T_{migliorabile}}{a} + T_{NON-migliorabile}} = \frac{1}{\frac{p}{a} + (1-p)} = \frac{1}{\frac{p}{a} + 1-p}$$

S : speedup
 P_{dopo} : prestazioni dopo
 P_{prima} : prestazioni prima
 T_{prima} : tempo prima
 T_{dopo} : tempo dopo
 $T_{migliorabile}$: tempo migliorabile
 $T_{NON-migliorabile}$: tempo non migliorabile
 a : parte migliorabile di un fattore a dove il tempo si riduce di un fattore a
 p : parte del tempo 'migliorabile' = $T_{migliorabile}/T_{prima}$
 a : speed-up della parte di sistema migliorabile

parte che si è avvertita di un fattore f

Quindi: $0 \leq p \leq 1$
 Mentre: $a \geq 1$

$0 \leq p \leq 1$
 $a < 1$, peggioro
 $a > 1$, va a migliorare
 $p=1$ se si migliora tutto il sistema
 $p=0$ non si migliora niente

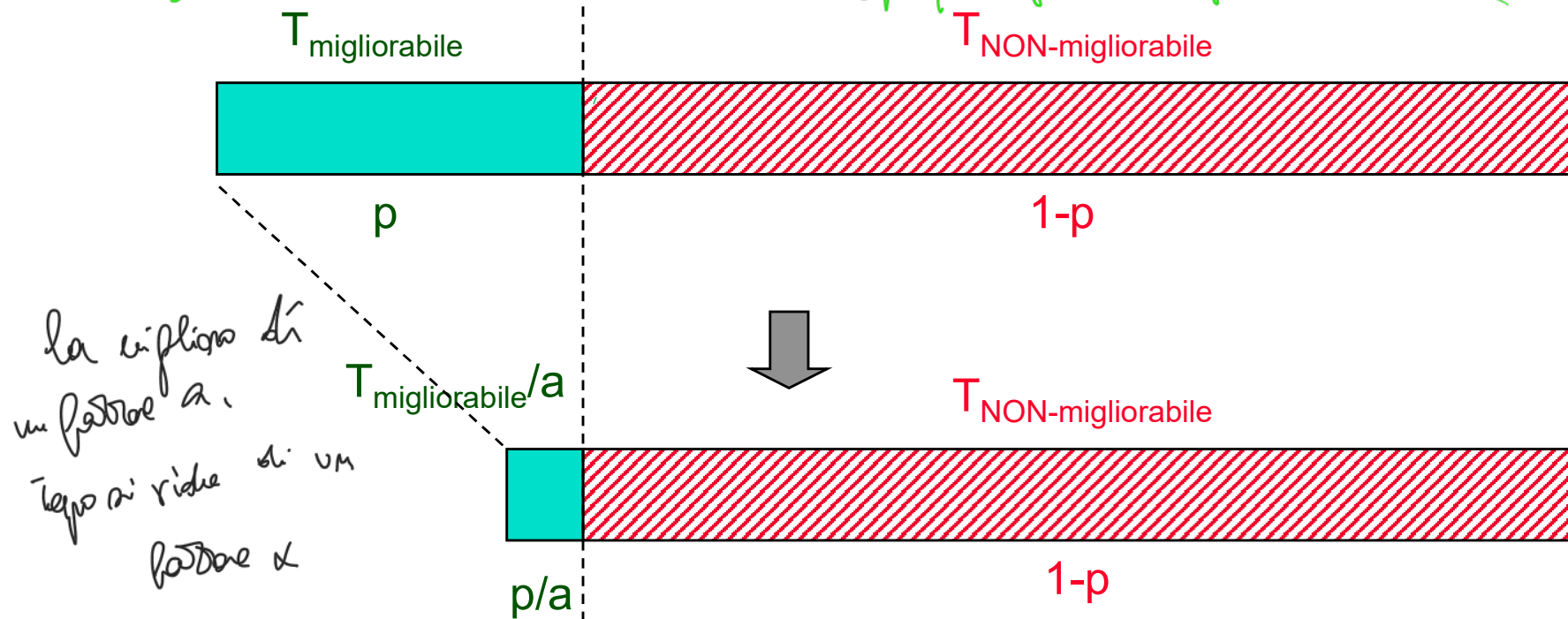
$p=0$ non migliora niente
 $p=1$ migliora l'intero sistema

Legge di Amdhal (2)

Dimostrazione
“grafica” della
legge di Amdhal

$$S = \frac{1}{\frac{p}{a} + (1-p)}$$

Tempo per eseguire programma



p = parte del tempo 'migliorabile' = $T_{\text{migliorabile}} / T_{\text{prima}}$

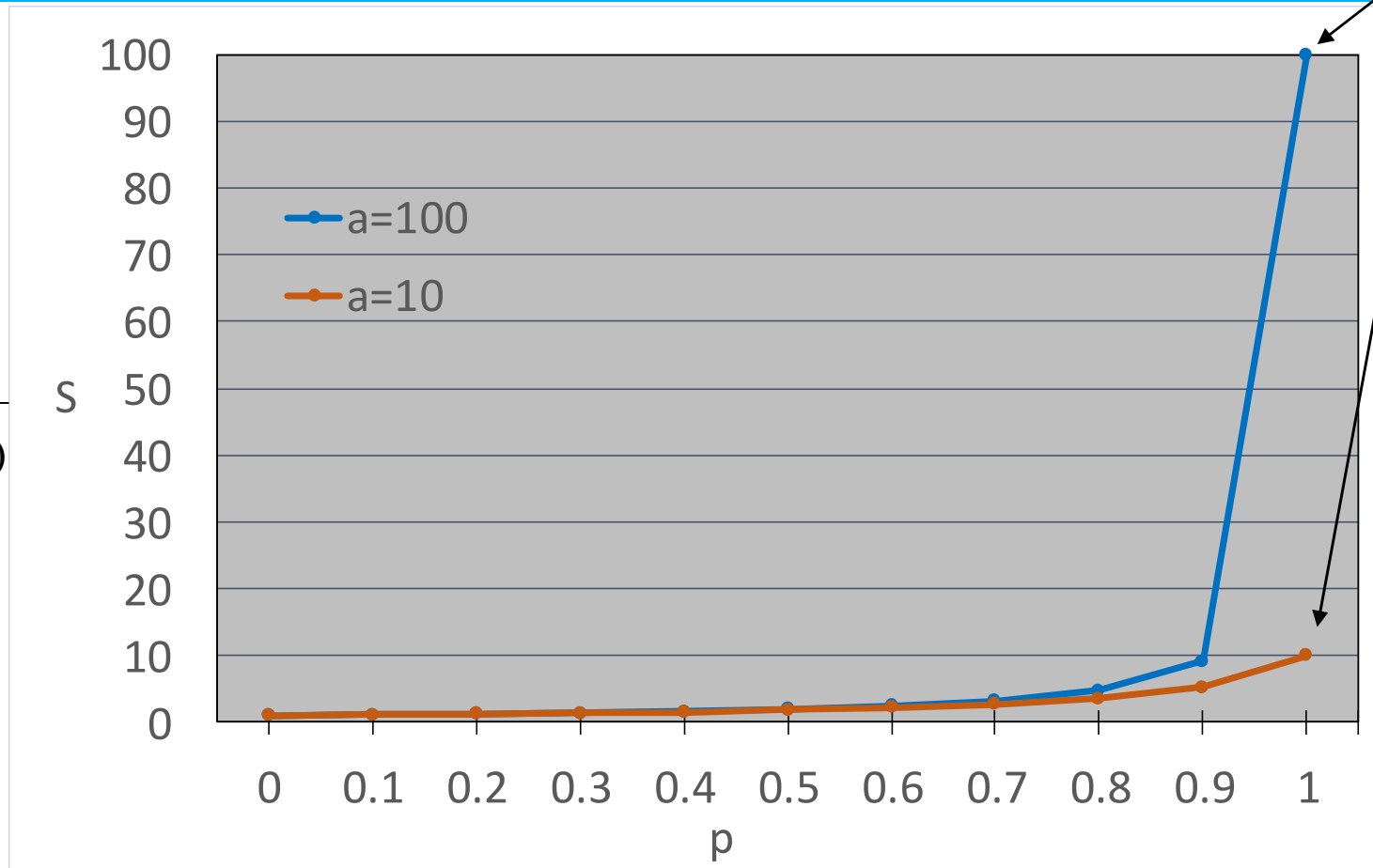
a = speed-up della parte di sistema migliorabile

Legge di Amdhal (3) - S al variare di p

fissato a

Nota: questo punto
(per p=1) vale 'a'
è una costante

$$S = \frac{1}{\frac{p}{a} + (1-p)}$$



→ lo speed-up di prestazioni S assume valori significativamente alti solo quando la parte migliorata p è abbastanza grande (es. p>0.8)

Se parte che
migliora piccola
non migliora
velocità

→ è importante concentrare i miglioramenti su una parte del sistema che sia consistente, 80,8%.

Per avere uno speed-up consistente
grosse del sistema

devo migliorare parti

Legge di Amdhal (4) - S al variare di a

Come varia S variando a

- Supponiamo che il 90% ($\rightarrow p=0.9$) di un programma possa essere eseguita in parallelo (es. aggiungendo più processori)

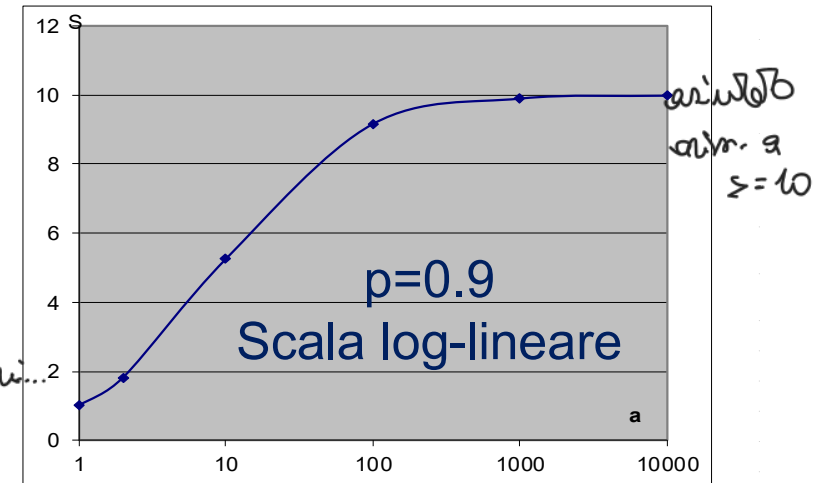
$$S = \frac{1}{\frac{p}{a} + (1-p)}$$

$$\lim_{a \rightarrow \infty} S = \frac{1}{1-p}$$

$\rightarrow$ lo speed-up di prestazioni S ottenibile migliorando una parte p è limitato da $1/(1-p)$

a	S	p
1	1.000	0.9
2	1.818	0.9
10	5.263	0.9
100	9.174	0.9
1000	9.911	0.9
10000	9.991	0.9

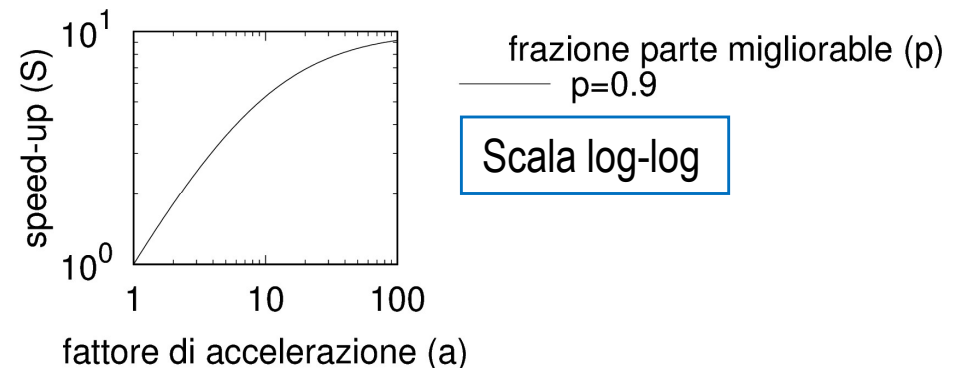
Se aggiungo processori...



Nell'esempio in figura, per $a \rightarrow \infty$ il massimo speed-up ottenibile è pari a $1/0.1=10$. Quindi invece di aggiungere 9999 processori conviene parallelizzare anche una parte del restante 10%

Nota:

Se faccio lo stesso disegno in scala log-log:



Conclusioni dalla legge di Amdhal

1) è molto importante migliorare la parte più utilizzata del sistema!

e deve essere consistente

2) A un certo punto è importante anche migliorare la parte meno utilizzata del sistema...
(ovvero spingere il più possibile p a valori molto alti)

2) dobbiamo dedicarci anche al resto